Installing Mono 2.10.9 on AFS

Friday, August 31, 2012
8:03 PM

Log out of AFS and log back in. Then execute the following:

```
$ cd ~/public
$ mkdir mono-2.4.3.1-4.e16
$ cd mono-2.4.3.1-4.e16
$ yumdownloader mono-core
$ rpm2cpio mono-core-2.4.3.1-4.el6.x86_64.rpm | cpio -idv
$ cd usr/bin
```

Note that at this point we have all the front-ends for Mono. The only two binary files in this directory are mono and monodir, and the other files are just short shell scripts that try to call /usr/bin/mono and /usr/bin/monodir, then redirect to the correct .exe under whatever location is returned my /usr/bin/monodir. In this case, ./monodir returns /usr/lib64/mono.

```
$ rm monodir
$ cat > monodir
#!/bin/sh
BASEDIR=`readlink -f $0 | xargs dirname`
echo ${BASEDIR}/../lib64/mono
^D
$ chmod +x monodir
$ vim -p `ls * | grep -v -P '^mono$' | grep -v -P '^monodir$' | grep -v -P '^mono-test-install'`
```

At this point, edit the 13 files so that each has the line

BASEDIR=`readlink -f $0 | xargs dirname`

just after the she-bang line and so that each uses ${BASEDIR}/mono and ${BASEDIR}/monodir instead of /usr/bin/mono and /usr/bin/monodir. A quick way to accomplish this is to insert the assignment to BASEDIR and then use the command

:%s/\/usr\/bin/${BASEDIR}/g

in each file.
As an example, here is the final gmcs:

```
#!/bin/sh
BASEDIR=`readlink -f $0 | xargs dirname`
exec ${BASEDIR}/mono $MONO_OPTIONS `${BASEDIR}/monodir`/2.0/gmcs.exe "$@"
```

Now back to commands:

```
$ set path = ( `pwd` $path )
$ cd ~/public
$ mkdir mono210
$ wget http://download.mono-project.com/sources/mono/mono-2.10.9.tar.bz2
$ tar xvjf mono-2.10.9.tar.bz2
$ cd mono-2.10.9
$ ./configure --prefix=`readlink -f ~`/public/mono210
$ make
$ make install
$ vim ~/.cshrc
```

Now there should be a working copy of Mono 2.10.9 in ~/public/mono210. The final step to use Mono is to add the following line to the .cshrc file:

set path = ($path `readlink -f ~`/public/mono210/bin)

To test the configuration, log out and log back in to AFS, and then try the following:

```
$ cd ~/public
$ mkdir test
$ cd test
$ cat > Test.cs
using System;
using System.Linq;

namespace Foo
{
class Bar
{
```

```
    static void Main(string[] args)
    {
        if ((new[] { 0, 1, 2 }).Any())
            PrintIfNotNull(s: "Hello, World!");
    }

    static void PrintIfNotNull(string s = null)
    {
        if (s != null)
            Console.WriteLine("{0}", s);
    }
}
}
^D
$ dmcs Test.cs
$ mono Test.exe
$ cd ..
$ rm -r test
```

Congratulations on installing Mono!

Resources that I used for these instructions:

- Getting an RPM using yumdownloader: http://superuser.com/a/303084 or http://superuser.com/questions/303083/how-to-extract-rpm-from-rpm-database-in-redhat
- Unpacking an RPM to a local directory: http://superuser.com/a/210055 or http://superuser.com/questions/209808/how-can-i-install-an-rpm-without-being-root
- Getting the directory name where a shell script is located: http://stackoverflow.com/a/1638397 or http://stackoverflow.com/questions/242538/unix-shell-script-find-out-which-directory-the-script-file-resides
- Downloading Mono source code: http://download.mono-project.com/sources/mono/

Installing F# 2.0 on AFS

Saturday, September 1, 2012
10:58 AM

First, install Mono 2.10.9 on AFS. Then execute the following:

```
$ cd ~/public
$ mkdir fsharp20
$ mkdir fsharp-install
$ cd fsharp-install
$ git clone git://github.com/fsharp/fsharp.git
$ cd fsharp
$ setenv PKG_CONFIG_PATH `readlink -f ~`/public/mono210/lib/pkgconfig
```

Adding ~/public/mono210/lib/pkgconfig to the path allows the autogen script to determine where Mono is installed.

```
$ ./autogen.sh --prefix=`readlink -f ~`/public/fsharp20
$ make
$ make install
$ cd ~/public
$ cat > gacassemblies.txt
fsharp20/lib/mono/gac/FSharp.Core/2.0.0.0__b03f5f7f11d50a3a/FSharp.Core.dll
fsharp20/lib/mono/gac/FSharp.Core/4.0.0.0__b03f5f7f11d50a3a/FSharp.Core.dll
fsharp20/lib/mono/gac/FSharp.Build/2.0.0.0__b03f5f7f11d50a3a/FSharp.Build.dll
fsharp20/lib/mono/gac/FSharp.Build/4.0.0.0__b03f5f7f11d50a3a/FSharp.Build.dll
fsharp20/lib/mono/gac/FSharp.Compiler/2.0.0.0__b03f5f7f11d50a3a/FSharp.Compiler.dll
fsharp20/lib/mono/gac/FSharp.Compiler/4.0.0.0__b03f5f7f11d50a3a/FSharp.Compiler.dll
fsharp20/lib/mono/gac/FSharp.Compiler.Interactive.Settings/2.0.0.0__b03f5f7f11d50a3a/FSharp.Compiler.Interactive.Settings.dll
fsharp20/lib/mono/gac/FSharp.Compiler.Interactive.Settings/4.0.0.0__b03f5f7f11d50a3a/FSharp.Compiler.Interactive.Settings.dll
fsharp20/lib/mono/gac/FSharp.Compiler.Server.Shared/2.0.0.0__b03f5f7f11d50a3a/FSharp.Compiler.Server.Shared.dll
fsharp20/lib/mono/gac/FSharp.Compiler.Server.Shared/4.0.0.0__b03f5f7f11d50a3a/FSharp.Compiler.Server.Shared.dll
^D
$ gacutil -check_refs -il gacassemblies.txt
$ rm gacassemblies.txt
$ vim ~/.cshrc
```

Now there should be a working copy of F# 2.0 in ~/public/fsharp20. The final step to use F# is to add the following line to the .cshrc file:

```
set path = ( $path `readlink -f ~`/public/fsharp20/bin )
```

To test the configuration, log out and log back in to AFS, and then try the following (you can write to file Test.fs instead of Test.fsx - I have yet to find a difference):

```
$ cd ~/public
$ mkdir test
$ cd test
$ cat > Test.fsx
#light

open System

[<EntryPoint>]
let main args =
    printfn "Hello, World!"
    0
^D
$ fsharpc Test.fsx
$ mono Test.exe
$ fsharpi
$ cd ..
$ rm -r test
```

Congratulations on installing F#!

Resources that I used for these instructions:

- Installing F# on Linux: http://fsxplat.codeplex.com/wikipage?title=FSharpLinux

Installing the F# PowerPack on AFS

Sunday, September 2, 2012
12:14 AM

N.B. This solution is inelegant, because it compiles assemblies but does not give them a strong name, and relies on MONO_PATH, a discouraged practice.

First, install F# 2.0 on AFS. To build the assemblies, execute the following:

```
$ cd ~/public
$ git clone git://github.com/fsharp/powerpack.git
$ mkdir powerpack/bin
$ cd powerpack/src/FSharp.Compiler.CodeDom
$ fsharpc -a -o FSharp.Compiler.CodeDom.dll ../assemblyinfo.Common.fs
assemblyinfo.FSharp.Compiler.CodeDom.dll.fs ../FSharp.PowerPack/CompilerLocationUtils.fs codedomvisitor.fs generator.fs
compiler.fs codeprovider.fsi codeprovider.fs
$ mv FSharp.Compiler.CodeDom.dll ../../bin
$ cd ../FSharp.PowerPack
$ fsharpc -a -o FSharp.PowerPack.dll ../assemblyinfo.Common.fs AssemblyInfo.fs StructuredFormat.fsi StructuredFormat.fs
math/q.fsi math/q.fs ResizeArray.fsi ResizeArray.fs HashMultiMap.fsi HashMultiMap.fs AsyncOperations.fsi AsyncOperations.fs
AsyncWorker.fsi AsyncWorker.fs AsyncStreamReader.fsi AsyncStreamReader.fs HashSet.fsi HashSet.fs TaggedCollections.fsi
TaggedCollections.fs TaggedHash.fsi TaggedHash.fs Measure.fsi Measure.fs SI.fs PhysicalConstants.fs Lazy.fsi Lazy.fs
Permutation.fsi Permutation.fs math/INumeric.fsi math/INumeric.fs math/complex.fsi math/complex.fs math/associations.fsi
math/associations.fs math/matrix.fsi math/matrix.fs NativeArray.fsi NativeArray.fs math/NativeArrayExtensions.fsi
math/NativeArrayExtensions.fs Lexing.fsi Lexing.fs Parsing.fsi Parsing.fs Arg.fsi Arg.fs LazyList.fsi LazyList.fs
$ mv FSharp.PowerPack.dll ../../bin
$ cd ../FSharp.PowerPack.Compatibility
$ fsharpc -a -o FSharp.PowerPack.Compatibility.dll -I `readlink -f ~`/public/powerpack/src/FSharp.PowerPack -r
FSharp.PowerPack.dll ../assemblyinfo.Common.fs assemblyinfo.fs lazy.fsi lazy.fs Compat.String.fsi Compat.String.fs
Compat.List.fsi Compat.List.fs Compat.Array.fsi Compat.Array.fs Compat.Array2D.fsi Compat.Array2D.fs Compat.Seq.fsi
Compat.Seq.fs byte.fsi byte.fs char.fsi char.fs sbyte.fs int16.fs int32.fsi int32.fs int64.fsi int64.fs uint32.fsi uint32.fs
uint64.fsi uint64.fs float.fsi float.fs hashtbl.fsi hashtbl.fs arg.fsi arg.fs sys.fsi sys.fs obj.fsi obj.fs filename.fsi
filename.fs map.fsi map.fs set.fsi set.fs printexc.fsi printexc.fs pervasives.fsi pervasives.fs buffer.fsi buffer.fs
lexing.fsi lexing.fs parsing.fsi parsing.fs
$ mv FSharp.PowerPack.Compatibility.dll ../../bin
$ cd ../FSharp.PowerPack.Linq/
$ fsharpc -a -o FSharp.PowerPack.Linq.dll ../assemblyinfo.Common.fs assemblyinfo.FSharp.PowerPack.Linq.dll.fs
FuncConvertExtensions.fsi FuncConvertExtensions.fs Linq.fsi Linq.fs LinqQueries.fsi LinqQueries.fs Assembly.fs
$ mv FSharp.PowerPack.Linq.dll ../../bin
$ cd ../FSharp.PowerPack.Parallel.Seq/
$ vim pseq.fsx
```

Just change the load line (line 3) to all lower-case i.e. #load "pseq.fsx" rather than #load "Pseq.fsx".

```
$ fsharpc -a -o FSharp.PowerPack.Parallel.Seq.dll ../assemblyinfo.Common.fs assemblyinfo.FSharp.PowerPack.Parallel.Seq.dll.fs
pseq.fsi pseq.fs Assembly.fs pseq.fsx
$ mv FSharp.PowerPack.Parallel.Seq.dll ../../bin/
$ cd ../FSharp.PowerPack.Metadata
$ fsharpc -a -o FSharp.PowerPack.Metadata.dll ../assemblyinfo.Common.fs assemblyinfo.FSharp.PowerPack.Metadata.dll.fs
Prelude.fs FSComp.fs FlatList.fs QueueList.fs PrettyNaming.fs il.fs tast.fs env.fs tastops.fs
pickle.fs ../FSharp.PowerPack/CompilerLocationUtils.fs Metadata.fsi Metadata.fs
$ mv FSharp.PowerPack.Metadata.dll ../../bin/
$ vim ~/.cshrc
```

Now, to reference these in the future, we need to use the -I compiler flag to reference `readlink -f ~`/public/powerpack/bin, and we need to add the following line to the .cshrc in order to use the assemblies at runtime:

    setenv MONO_PATH `readlink -f ~`/public/powerpack/bin

We don't yet have fslex and fsyacc. We can't compile from source because these depend on Fsharp.PowerPack.Build.Tasks, which depends on some MSBuild DLLs. So we get these from the official binary distribution. Download
the F# PowerPack from http://fsharppowerpack.codeplex.com/releases/view/45593 and upload to AFS at ~/public/FSharpPowerPack.zip. Then execute the following:

```
$ cd ~/public
$ unzip FSharpPowerPack.zip
$ vim ~/.cshrc
```

The fslex and fsyacc executables are in this directory, so for convenience we add the following to the .cshrc file:

    alias fslex 'mono ~/public/FSharpPowerPack-2.0.0.0/bin/fslex.exe'
    alias fsyacc 'mono ~/public/FSharpPowerPack-2.0.0.0/bin/fsyacc.exe'

Log out and log back in to AFS. To test, use the following:

```
$ cd ~/public
$ mkdir test
$ cd test
$ cat > test.fsx
#light

open Microsoft.FSharp.Linq.QuotationEvaluation

[<EntryPoint>]
let main args =
    let adderExpr = <@ fun i -> i + 1 @>.ToLinqExpression()
    let adder = <@ fun i -> i + 1 @>.Compile()
    printfn "Hello, World!"
```

```
    0
^D
$ fsharpc -I `readlink -f ~`/public/powerpack/bin -r FSharp.PowerPack.Linq.dll test.fsx
$ mono test.exe
$ git clone git://github.com/fsharp/powerpack.git
$ cd powerpack/workyard/tests/LexAndYaccMiniProject
$ fslex --unicode Lexer.fsl
$ fsyacc --module Parser Parser.fsy
$ fsharpc -o LexAndYaccMiniProject.exe -I ~/public/powerpack/bin/ -r FSharp.PowerPack.dll Parser.fsi Parser.fs Lexer.fs
Program.fs
$ mono LexAndYaccMiniProject.exe
$ cd ~/public
$ rm -rf test
```

Congratulations on installing the F# PowerPack!

References:

- Why not to use MONO_PATH: http://www.mono-project.com/Assemblies_and_the_GAC
- The difference between a module and an assembly in .NET: http://stackoverflow.com/questions/9271805/net-module-vs-assembly
- The original prompt for compiling the F# PowerPack from source - the download is for .NET 2.0: http://stackoverflow.com/questions/5447539/problem-with-f-powerpack-method-not-found-error
- The source of the test program: http://stackoverflow.com/questions/6206406/how-to-install-and-use-f-powerpack-in-mono
    - A related post about DLL references by the same question asker: http://stackoverflow.com/questions/6207270/referencing-net-dll-when-compiliationwith-mono
        - This post is the source of "the GAC and MONO_PATH don't matter to the compiler".
- Documentation for .NET/Mono builtin tools and signing:
    - How to sign an assembly with a strong name: http://msdn.microsoft.com/en-us/library/xc31ft41(v=vs.100).aspx
    - Delay signing an assembly: http://msdn.microsoft.com/en-us/library/t07a3dye(v=vs.100).aspx
    - al: http://msdn.microsoft.com/en-us/library/c405shex(v=vs.100).aspx
    - sn: http://msdn.microsoft.com/en-us/library/k5b5tt23.aspx
    - An old MSDN Mag article about strong names: http://msdn.microsoft.com/en-us/magazine/cc163583.aspx
- Forum answers related to signing:
    - Problem signing a DLL in Mono: http://stackoverflow.com/questions/3733148/problems-signing-a-dll-in-f-on-mono, especially the second answer
        - This solution doesn't actually work, because sn -R requires that the assembly already be signed with the same private key, and Mono doesn't implement sn -Vr, which is required to make sn temporarily remove this restriction.